CSC 317: Data Structures and Algorithm Analysis

Dilip Sarkar

Department of Computer Science
University of Miami

Outline I

Dictionary Operations

- Let $X = \{X_1, X_2, \cdots, X_n\}$ be a set of n objects
- Assume each object X_i has an associated unique key $X_i.key$
- Let the universe of these keys be U, where $U = \{0, 1, \cdots, m-1\}$
- Example: 8-digit UM student ID numbers, $|U| = m = 10^8$
- We want to perform the following three **dictionary** operations on the table T
  - SEARCH(T,k)
    return $T[k]$
  - INSERT(T,x)
    $T[x.key] = x$
  - DELETE(T,x)
    $T[x.key] = NIL$

IMPLEMENTATION of THESE OPERATIONS

Data Structure	Operations		
	Find	Insert	Delete
Stack	$O(n)$	$O(1)$	$O(n)$
Queue	$O(n)$	$O(1)$	$O(n)$
Linked List (unsorted)	$O(n)$	$O(1)$	$O(n)$
Linked List (sorted)	$O(n)$	$O(n)$	$O(n)$
Array (unsorted)	$O(n)$	$O(1)$	$O(n)$
Array (sorted)	$O(\lg n)$	$O(n)$	$O(n)$

- Can we use array?
- Yes, only if m is small.
- Why?

Direct-Address Tables

- If m is small, we can keep all of them in a table T
  - DIRECT-ADDRESS-SEARCH(T,k)
    return $T[k]$
  - DIRECT-ADDRESS-INSERT(T,x)
    $T[x.key] = x$
  - DIRECT-ADDRESS-DELETE(T,x)
    $T[x.key] = NIL$
  - If m is small, we can keep one item at one location.

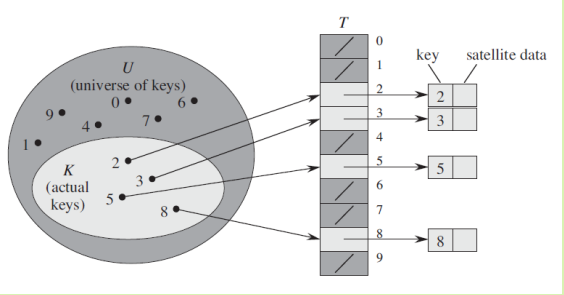

Figure: Implementation of Direct-Access Table of size 10.

- Each of the three operations can be done in constant time
  - SEARCH takes $O(1)$ time
  - INSERT takes $O(1)$ time
  - DELETE takes $O(1)$ time
- If size of the table is k, we can keep only $0, 1, \cdots, (k-1)$ elements
- Suppose we have less than k elements, can we keep them in the table?
- Maybe not?
- Because for some elements X_i, $X_i.key > k$
- Can we find a function $h(x)$ such that $h(X_i.key) = j$ for $i \leq (m-1)$?
- Yes. But?

Hash Tables

- **Collinsion:** For any pair of keys, if $h(X_i.key) = h(X_j.key)$, where $i \neq j$,
- Figure below shows $h(k_2) = h(k_5)$

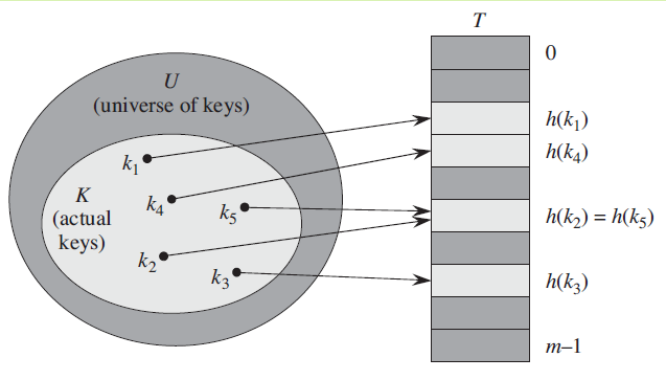

Figure: Curse of Collion.

- Mapping $h(x)$ creates with k_2 and k_5
- The functions used for mapping keys are called *hash functions*.
- Example 1: $h(k) = k \mod 10$ that will map all integers to $\{0, 1, \cdots, 9\}$.
  - $h(15) = 5, h(11) = 1, h(25) = 5,$ and $h(111) = 1$
- In fact, this mapping keeps only the last digit of a given non-negative integer
- Example 2: $h_2(k) = d_i$, where d_i is the ith digit of the integer k
- Example 3: $h_3(s) = s_i$, where s_i is the ith character of the string s.

Collision Resolution by Chaining

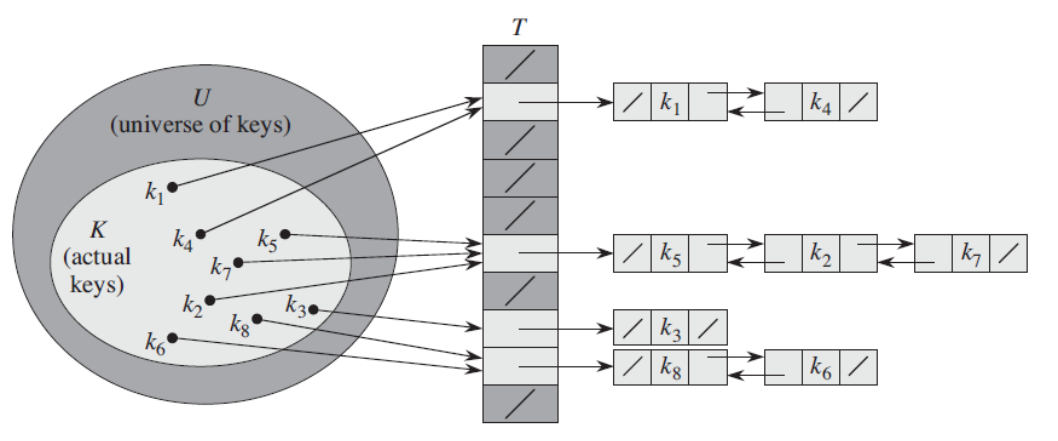

Figure: Collision Resolution By Chaining

DICTIONARY Operations

- Chained-Hash-Inset (T, x)
  insert x at the head of the list $T[h(x.key)]$
- Chained-Hash-Search (T, k)
  search for an element with key k in the list $T[h[k]$
- Chained-Hash-Delete (T, x)
  delete x from the list $T[h(x.key)]$

Analysis of Hashing with Chaining I

- Assume table T has m slots and n elements.
- Define **load** factor: $\alpha = n/m$
- In the worst case all n data can map at one location
- Worst case performance $\Theta(n)$
- What is average-case performance?
- Depends on how the hash function distributes the set of keys to m slots
- **Simple uniform hashing:** given an element x, it is hashed to any of the m slots with equal probability.
- Let the length of the list at $T[j]$ be n_j
- Thus, $n = n_0 + n_1 + \cdots + n_{m-1}$
- Expected value of $E(n_j) = n/m = \alpha$

Theorem (11.1)

In a hash table in which collision are resolved by chaining, an **unsuccessful search** *search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.*

Proof.

Computing $h(k)$ takes $O(1)$ time.

How many steps an unsuccessful search at location $h(k)$ will take?

Number of elements at location $T[h(k)]$

Expected length $E(n_{h(k)})$
$= \frac{1}{m}(n_0 + n_1 + \cdots + n_{m-1}) = \frac{n}{m} = \alpha$

□

Analysis of Hashing with Chaining II

Theorem (11.2)

In a hash table in which collision are resolved by chaining, a **successful search** *search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.*

Proof.

Computing $h(k)$ takes $O(1)$ time.

How many steps a successful search at location $h(k)$ will take?

A harder question: we will use indicator random valuable. $X_{ij} = I\{h(k_i) = h((k_j))\}$ for keys of x_i and x_j to show average-case complexity $\Theta(1+\alpha)$.

□

We are looking for x_i which has been inserted at $T(h(k_i))$

But how many elements are at $T(h(k_i))$?

One plus all elements after x_i that hashed at this location.

$= 1 + \sum_{j=i+1}^{n} E[X_{ij}]$

$= 1 + \sum_{j=i+1}^{n} \frac{1}{m}$

Now, we must sum for $i = 1$ to n

$\Rightarrow E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} X_{ij}\right)\right]$

$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} E[X_{ij}]\right)$

$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} \frac{1}{m}\right)$

$= 1 + \frac{1}{nm}\sum_{i=1}^{n}(n-i)$

Analysis of Hashing with Chaining III

Theorem (11.2)

In a hash table in which collision are resolved by chaining, a **successful search** *takes average-case time* $\Theta(1+\alpha)$*, under the assumption of simple uniform hashing.*

Proof.

Computing $h(k)$ takes $O(1)$ time.

How many steps a successful search at location $h(k)$ will take?

A harder question: we will use indicator random valuable. $X_{ij} = I\{h(k_i) = h((k_j))\}$ for keys of x_i and x_j to show average-case complexity $\Theta(1+\alpha)$.

□

Now, we must sum for $i = 1$ to n

$= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)$

$= \frac{1}{n}\sum_{i=1}^{n}1+\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=i+1}^{n}1\right)$

$= 1+\frac{1}{nm}\sum_{i=1}^{n}\left(\sum_{j=i+1}^{n}1\right)$

Because $\sum_{j=i+1}^{n}1 = \sum_{j=1}^{n-i}1 = n-i$

We have $1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)$

$1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right)$

$= 1+\frac{1}{nm}\left(\frac{2n^2-(n^2+n)}{2}\right)$

$= 1+\frac{n-1}{2m} = 1+\frac{\alpha}{2}-\frac{\alpha}{2n}$

Including time for a hash, expected total time is,

$1+(1+\frac{\alpha}{2}-\frac{\alpha}{2n}) = \Theta(1+\alpha)$

Hash Functions

- What is a good hash function?
  - Simple uniform hashing for the given set of keys
  - Unless we know the distribution of keys, above property may not be satisfied
  - Fast computation time
- Some commonly used hash functions
  - **The division method**: $h(k) = k \mod m$
  - **The multiplication method:** $h(k) = \lfloor m(kA \mod 1)\rfloor$ for a constant $0 < A < 1$.
  - **Universal hashing:** Let $H = \{h_1(k), h_2(k), \cdots$ be a collection of hash functions. Select one randomly for each time a hashing is necessary.
  - Let p be a prime number, $1 < a < p$, and $1 \le b < p$
  - A class of universal hash functions: $h_{a,b}(k) = ((ak+b) \mod p) \mod m$
  - Example: for $a = 3, b = 4, p = 17$ and $m = 6$

    $h_{3,4}(8) = ((3\times 8+4) \mod 17) \mod 6$

    $= ((24+4) \mod 17) \mod 6$

    $= (28 \mod 17) \mod 6$

    $= 11 \mod 6 = 5$

Collision Resolution by Open Addressing I

- **Open Addressing:** All elements are stored in the table, *no external linked list*.
- If the location computed by **first** hashing is occupied,

  continue computing **different locations** using a *secondary* hash function,

  until an unoccupied location is found. For $i = \{1, 2, \cdots\}$
- Some commonly used secondary (also known as rehash function).
  - **Linear proving:** $h(k, i) = (h(k) + c_1 i) \mod m)$
  - **Quadratic Proving:** $h(k, i) = (h(k) + c_1 i + c_2 i^2) \mod m)$
  - **Double hashing:** $h(k, i) = (h_1(k) + i \times h_2(k)) \mod m)$

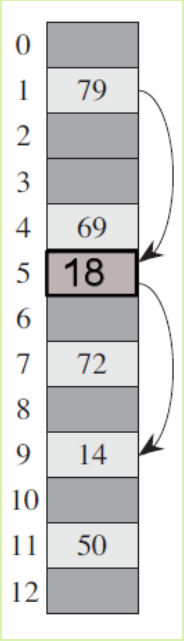

Figure: Collision resolution by open addressing

Collision Resolution by Open Addressing II

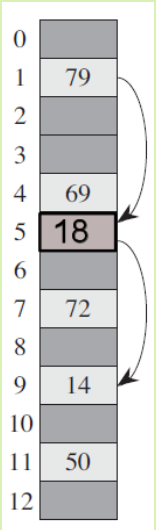

Figure: Collision resolution by open addressing

Let input key sequence be 79, 69, 72, 18, 14, and 50

$h_1(k) = k \mod 13$, $h_2(k) = 1 + (k \mod 11)$, and
$h(k, i) = (h_1(k) + i \times h_2(k)) \mod 13$

$h(k, 0) = (h_1(79) + 0 \times h_2(79)) \mod 13 = 1$, key 79 is inserted at T[1]

$h(k, 0) = (h_1(69) + 0 \times h_2(69)) \mod 13 = 4$, key 69 is inserted at T[4]

$h(k, 0) = (h_1(72) + 0 \times h_2(72)) \mod 13 = 7$, key 72 is inserted at T[7]

$h(k, 0) = (h_1(18) + 0 \times h_2(18)) \mod 13 = 5$, key 18 is inserted at T[5]

$h(k, 0) = (h_1(14) + 0 \times h_2(14)) \mod 13 = 1$, T[1] is occupied, a collision!

Because $h_2(14) = 1 + 14 \mod 11 = 1 + 3 = 4$

$h(k, 0) = (h_1(14) + 1 \times 4) \mod 13 = 5$, T[5] is occupied, another collision!

$h(k, 0) = (h_1(14) + 2 \times 4) \mod 13 = 9$, key 14 is inserted at at T[9]

Analysis of Hash Tables with open addressing I

Theorem (11.6)

Given an open address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$, assuming uniform hashing.

Proof.

There are n items in a table of size m.

We need to estimate $E(X) = \#$ of proves before we find an empty location (FALURE). X is a random variable.

$Pr(X = 1) = \frac{n}{m}$

Thus, $Pr(X = 1 \text{ and } X = 2) = Pr(X = 1) \times Pr(X = 2) = \frac{n}{m} \times \frac{n-1}{m-1}$

- We continue in the next column.

□

Note that:

$(\frac{n-i}{m-i} - \alpha) = (\frac{n-i}{m-i} - \frac{n}{m})$

$= (\frac{mn-im-mn+in}{m(m-i)}) = (\frac{i(n-m)}{m(m-i)}) < 0$

because $n < m$

$\Rightarrow \frac{n-i}{m-i} < \alpha$ for $m > n$

$Pr(X \geq i) = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+2}{m-i+2}$

$Pr(X \geq i) \leq \alpha^{(i-1)}$

$E(X) = \sum_{i=1}^{\infty} Pr(X \geq i)$

$= \sum_{i=1}^{\infty} \alpha^{(i-1)} = \frac{1}{1-\alpha}$ □

Analysis of Hash Tables with open addressing II

Corollary (11.7)

Inserting an element in an open-address hash table with load factor α requires at most $\frac{1}{1-\alpha}$ probes on average, assuming uniform hashing.

Proof.

Since, an element is inserted after an empty location is identified after an unsuccessful search, average number of proves is same as as an unsuccessful search., which i $\frac{1}{1-\alpha}$.

□

Analysis of Hash Tables with open addressing III

Theorem (11.8)

Given an open address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$, assuming uniform hashing and assuming that each key in the table is equally likely to be searched.

A search for a key k produces the same sequence as when the element was inserted. By Corollary 11.7, if k was the $(i+1)$th key inserted in the table, expected number of proves is at most, $\frac{1}{1-i/m)} = \frac{m}{m-i}$. Averaging over n keys, we get expected number of proves for a successful search.

Proof.

$$\begin{aligned}
\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\
&= \frac{1}{\alpha}\sum_{i=0}^{n-1}\frac{1}{m-i} \\
&= \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \\
&\leq \frac{1}{\alpha}\int_{m-n}^{n-1}\frac{1}{x}dx \\
&\qquad\qquad \text{by inequality (A.12)} \\
&\leq \frac{1}{\alpha}\ln\frac{m}{m-n} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}
\end{aligned}$$

□